Пермский филиал федерального государственного автономного
образовательного учреждения высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Виноградов Никита Андреевич

ОРГАНИЗАЦИЯ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ. СТРУКТУРНЫЙ ПАТТЕРН КОМПОНОВЩИК

Лабораторная работа

студента образовательной программы «Программная инженерия»
по направлению подготовки *09.03.04 Программная инженерия*

Руководитель
к.т.н., доцент кафедры Информационных технологий в бизнесе НИУ ВШЭ-Пермь

А.В. Кычкин

Пермь, 2020 год

Оглавление

Глава 1. Компоновщик

Это структурный паттерн проектирования, который позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, как будто это единичный объект.

1.1. Назначение

Компоновщик предлагает рассматривать через единый интерфейс с общим методом получения информации.

1.2. Структура

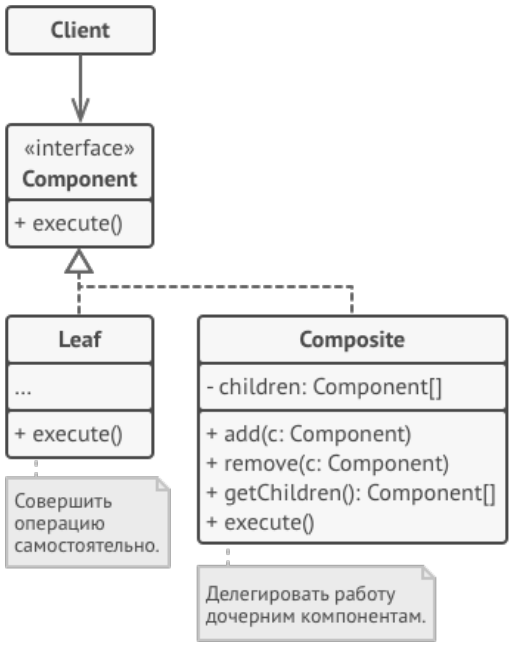

Рис. 1.1. Структура классов паттерна Компоновщик

Участники:

- *Client* - класс который вызывает методы компонента
- *Component* - интерфейс задает базу для остальных компонентов системы
- *Composite* - класс который выполняет работу над дочерними объектами в системе и позволяет пользоваться единым представлением

- *Leaf* - класс дочерние элементы компоненты на которых возлагается работа

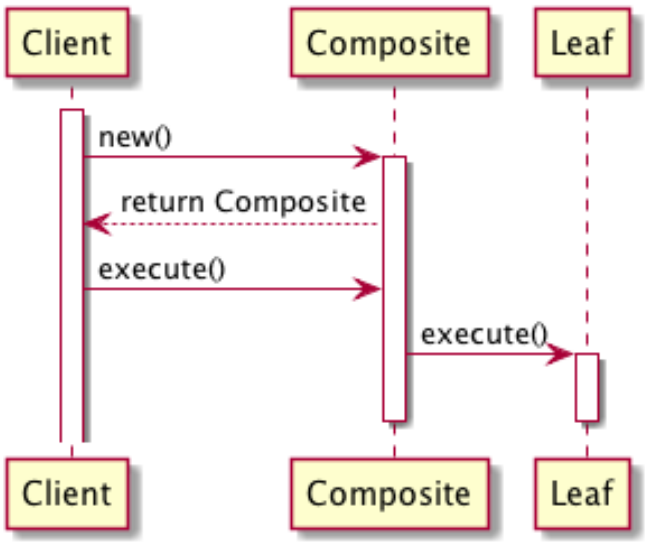

Рис. 1.2. Диаграмма последовательности паттерна Компоновщик

1.3. Способ применения

Данный паттерн необходим когда в системе если несколько уровней объектов, и необходимо делегировать работу для дочерних компонентам системы.

Глава 2. Реализация паттернов

2.1. Диаграмма классов

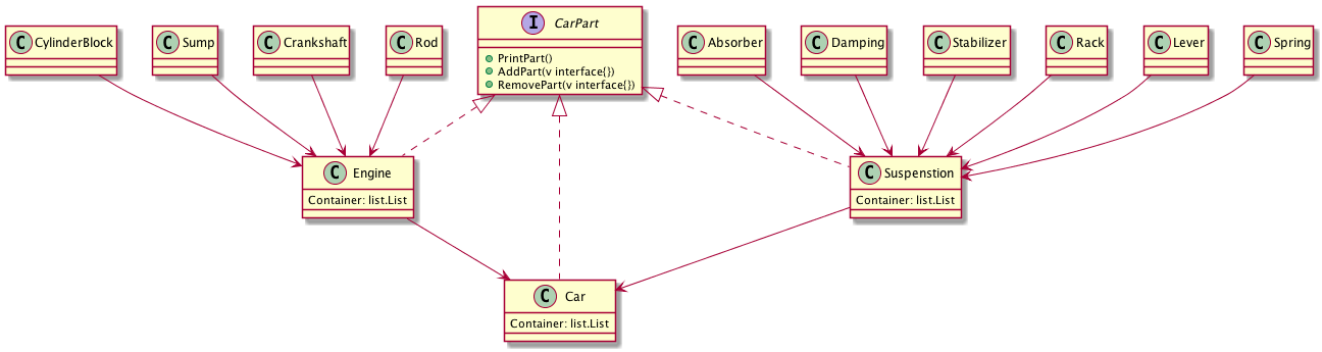

Рис. 2.1. Диаграмма классов паттерна Компоновщик

Участники:

- *Car* - главный класс состовляющий объект
- *CarPart* - интерфейс обобщающий дочерние элементы объектаы
- *Damping,Stabilizer,Rack,Lever,Spring,Absorber,CylinderBlock,Sump,Crankshaft,R* - классы выполняющие роль дочерних элементов системы.

2.2. Диаграмма последовательности

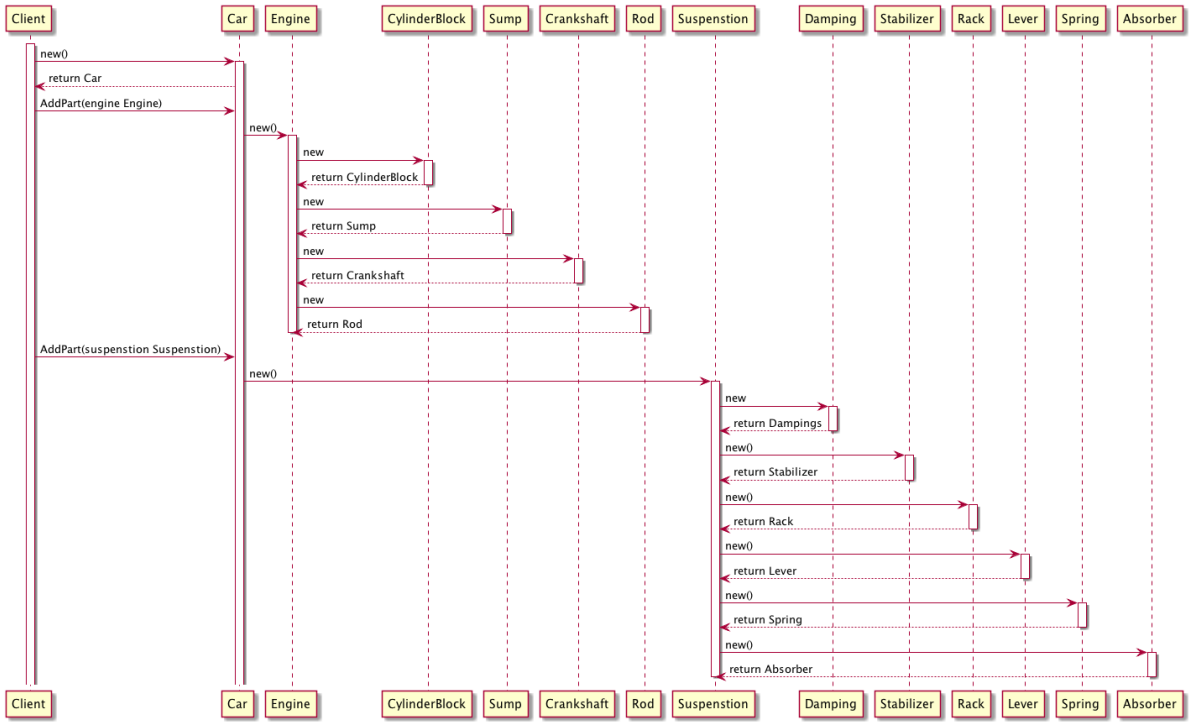

Рис. 2.2. Диаграмма последовательности паттерна Компоновщик

Участники:

- *Car* - главный класс состовляющий объект
- *CarPart* - интерфейс обобщающий дочерние элементы объектаы
- *Damping,Stabilizer,Rack,Lever,Spring,Absorber,CylinderBlock,Sump,Crankshaft,R* - классы выполняющие роль дочерних элементов системы

2.3. Код программы

```
package car

import (
"container/list"
"reflect"
)

func TypeFlex(v interface{}) string {
```

```
return reflect.TypeOf(v).String()
}

type Car struct {
Container *list.List
}

func (e Car) Print(num int) {
for e := e.Container.Front(); e != nil; e = e.Next() {

switch str := e.Value.(type) {
case CartPart:
println(TypeFlex(e.Value))


str.Print(num + 1)
break
}
}
}
func (e Car) AddPart(part interface{}) {

e.Container.PushBack(part)
println(e.Container.Len())
}
func (e Car) RemovePart(part *interface{}) {
el := list.Element{Value: part}
e.Container.Remove(&el)
}

type CartPart interface {
Print(num int)
}
package car
```

```
import "container/list"

//Демпфирующий
//Стабилизатор
//Стойка
//Рычаг
//Пружина
//Амортизатор

type Damping struct {
}
type Stabilizer struct {
}
type Rack struct {
}
type Lever struct {
}
type Spring struct {
}
type Absorber struct {
}
type Suspenstion struct {
Container *list.List
}

func (e Suspenstion) Print(num int) {
for e := e.Container.Front(); e != nil; e = e.Next() {
for i := 0; i < num; i++ {
print("-")
}
println(TypeFlex(e.Value))
}
}
```

```
func (e Suspenstion) AddPart(part interface{}) {
e.Container.PushBack(part)

}
func (e Suspenstion) RemovePart(part *interface{}) {
el := list.Element{Value: part}
e.Container.Remove(&el)
}

func NewSuspenstion() Suspenstion {
return Suspenstion{Container: list.New()}
}
package car

//блока цилиндровскартером ,
//
//- головкиблокацилиндров ,
//
//- поддонакартерадвигателя ,
//
//- поршнейскольцамиипальцами ,
//
//- шатунов,
//
//- коленчатоговала ,
//
//- маховика.
import (
"container/list"
)

type Engine struct {
Container *list.List
```

```
}

func (e Engine) Print(num int) {
for e := e.Container.Front(); e != nil; e = e.Next() {
for i := 0; i < num; i++ {
print("-")
}
println(TypeFlex(e.Value))
}
}

type CylinderBlock struct {
}
type Sump struct {
}
type Crankshaft struct {
}
type Rod struct {
}

func (e Engine) AddPart(part interface{}) {
e.Container.PushBack(part)
}
func (e Engine) RemovePart(part *interface{}) {
el := list.Element{Value: part}
e.Container.Remove(&el)
}

type EnginePart struct {
}
```